

Eliminating Unfounded Set Checking for HEX-Programs^{*}

Thomas Eiter, Michael Fink, Thomas Krennwallner, Christoph Redl, and Peter Schüller

Institut für Informationssysteme, Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria
{eiter, fink, tkren, redl, ps}@kr.tuwien.ac.at

Abstract. HEX-programs are an extension of the Answer Set Programming (ASP) paradigm incorporating external means of computation into the declarative programming language through so-called external atoms. Their semantics is defined in terms of minimal models of the Faber-Leone-Pfeifer (FLP) reduct. Developing native solvers for HEX-programs based on an appropriate notion of unfounded sets has been subject to recent research for reasons of efficiency. Although this has lead to an improvement over naive minimality checking using the FLP reduct, testing for foundedness remains a computationally expensive task. In this work we improve on HEX-program evaluation in this respect by identifying a syntactic class of programs, that can be efficiently recognized and allows to entirely skip the foundedness check. Moreover, we develop criteria for decomposing a program into components, such that the search for unfounded sets can be restricted. Observing that our results apply to many HEX-program applications provides analytic evidence for the significance and effectiveness of our approach, which is complemented by a brief discussion of preliminary experimental validation.

Keywords: Answer Set Programming, Nonmonotonic Reasoning, Unfounded Sets, FLP Semantics

1 Introduction

In the last years, Answer Set Programming (ASP) has emerged as an increasingly popular approach to declarative problem solving for a range of applications [2], thanks to expressive and efficient systems like *Smodels* [20], *DLV* [19], *cmodels* [17], and *CLASP* [15]. However, recent developments in computing, in which context awareness, distribution and heterogeneous information sources gain importance, raised the need for access to external sources in programs, be it in the context of the Web to access web services, databases, or ontological information in different formats, in the context of agents to acquire sensor input, etc.

To cater for this need, HEX-programs [11] extend ASP with so called external atoms, through which the user can couple any external data source with a logic program. Roughly, such atoms pass information from the program, given by predicate extensions,

^{*} This research has been supported by the Austrian Science Fund (FWF) project P20840, P20841, P24090, and by the Vienna Science and Technology Fund (WWTF) project ICT08-020.

into an external source which returns output values of an (abstract) function that it computes. This extension has been utilized for a range of applications, including querying data and ontologies on the Web, multi-context reasoning, and reasoning about actions and planning, to mention a few (cf. [5]). Notably, recursive data exchange between the rules and the external sources is supported, which makes the formalism powerful.

The semantics of a HEX-program Π is defined in terms of answer sets based on the FLP reduct [14]: an interpretation \mathbf{A} is an answer set of Π , if and only if it is a \subseteq -minimal model of the FLP-reduct $f\Pi^{\mathbf{A}}$ of Π wrt. \mathbf{A} , which is the set of all rules whose body is satisfied by \mathbf{A} . For ordinary logic programs, this semantics coincides with the one where the canonical GL-reduct [16] is in place of $f\Pi^{\mathbf{A}}$, and it is more appealing for extensions with nonmonotonic aggregates [14], and the more general external atoms in HEX-programs.

The evaluation of a HEX-program Π in the DLVHEX¹ solver proceeds in two steps as follows. In Step 1, external atoms are viewed as ordinary atoms (*replacement atoms*) and their truth values are guessed by choice rules that are added. The resulting ordinary ASP program $\hat{\Pi}$ is then evaluated by an ordinary ASP solver and each of its answer sets $\hat{\mathbf{A}}$ is checked against the external sources, i.e., the guess is verified. After that, the guess for the non-replacement atoms, called \mathbf{A} , is known to be a model of Π , and thus also of the reduct $f\Pi^{\mathbf{A}}$. Step 2 then checks whether \mathbf{A} is a \subseteq -minimal model or, equivalently, whether \mathbf{A} is unfounded-free [13], i.e., there exists no unfounded set (UFS) of Π wrt. \mathbf{A} .

Unfortunately, Step 2 is computationally expensive in general, and it is intractable even for Horn programs with nonmonotonic external atoms of polynomial complexity, as follows from results in [14]. It is thus worthwhile to be aware of cases where this test is tractable, or even better, superfluous such that Step 2 can be skipped.

Motivated by this issue, we consider in this paper programs Π for which the result of Step 1 is a \subseteq -minimal model of the reduct $f\Pi^{\mathbf{A}}$. We provide a sound *syntactic criterion* for deciding whether the minimality check is needed, and in further elaboration, we describe how a program can be *decomposed* into program components such that unfoundedness checks can be delegated to the components, and the necessity of Step 2 thus be assessed on a finer-grained level.

More in detail, our main contributions are the following:

- We present a *syntactic decision criterion* which can be used to decide whether a program possibly has unfounded sets. If the result of this check is negative, then the computationally expensive search for unfounded sets can be skipped. The criterion is based on atom dependency and, loosely speaking states that there are no cyclic dependencies of ground atoms through external atoms. This criterion can be efficiently checked for a given ground HEX-program using standard methods, and in fact applies to a range of applications, in particular, for input-stratified programs, where external sources are accessed in a workflow to produce input for the next stage of computation. However, there are relevant applications of HEX-programs where cycles through external atoms are essential, e.g., in encodings of problems on multi-context systems [1] or abstract argumentation systems [4], for which Step 2 cannot be skipped.
- In further elaboration, we consider a *decomposition* of a program Π into components based on the dependency graph that is induced by the program. We show that Π has

¹ <http://www.kr.tuwien.ac.at/research/systems/dlvhex/>

some unfounded set with respect to the candidate answer set \mathbf{A} if and only if (at least) one of the components Π_C in the decomposition has some unfounded set wrt. \mathbf{A} ; note that computing the decomposition is efficiently possible, and thus does not incur a large overhead. This allows us to apply the decision criterion for the necessity of Step 2 efficiently on a more fine-grained level, and the search for unfounded sets can be guided to relevant parts of the program. In particular, for the HEX-encoding of a Dung-style argumentation semantics [4] which we consider, the decomposition approach yields a considerable gain, as shown in a preliminary experimental evaluation.

This paper complements recent work on unfoundedness checking for HEX-programs in [7, 8], which is part of a larger effort to provide efficient evaluation of HEX-programs, based on new algorithms cf. [6]. By their wide applicability, our results are significant especially for many potential applications in practice.

2 Preliminaries

In this section, we start with some basic definitions, and then introduce syntax and semantics of HEX-programs and the notion of unfounded sets we are going to use.

A (*signed*) *literal* is a positive or a negative formula $\mathbf{T}a$ resp. $\mathbf{F}a$, where a is a ground atom of form $p(c_1, \dots, c_\ell)$, with predicate p and constants c_1, \dots, c_ℓ , abbreviated $p(\mathbf{c})$. For a literal $\sigma = \mathbf{T}a$ or $\sigma = \mathbf{F}a$, let $\bar{\sigma}$ denote its opposite, i.e., $\overline{\mathbf{T}a} = \mathbf{F}a$ and $\overline{\mathbf{F}a} = \mathbf{T}a$.

An *assignment* is a consistent set of literals $\mathbf{T}a$ or $\mathbf{F}a$, where $\mathbf{T}a$ expresses that $a \in \mathcal{A}$ and $\mathbf{F}a$ that $a \notin \mathcal{A}$. \mathcal{A} is *complete*, also called an *interpretation*, if no assignment $\mathbf{A}' \supset \mathbf{A}$ exists. We denote by $\mathbf{A}^{\mathbf{T}} = \{a \mid \mathbf{T}a \in \mathbf{A}\}$ and $\mathbf{A}^{\mathbf{F}} = \{a \mid \mathbf{F}a \in \mathbf{A}\}$ the set of atoms that are true, resp. false in \mathbf{A} , and by $\text{ext}(q, \mathbf{A}) = \{\mathbf{c} \mid \mathbf{T}q(\mathbf{c}) \in \mathbf{A}\}$ the extension of a predicate q . Furthermore, $\mathbf{A}|_q$ is the set of all literals over atoms of form $q(\mathbf{c})$ in \mathbf{A} . For a list $\mathbf{q} = q_1, \dots, q_k$ of predicates we write $p \in \mathbf{q}$ iff $q_i = p$ for some $1 \leq i \leq k$, and let $\mathbf{A}|_{\mathbf{q}} = \bigcup_j \mathbf{A}|_{q_j}$.

A *nogood* is a set $\{L_1, \dots, L_n\}$ of literals L_i , $1 \leq i \leq n$. An interpretation \mathbf{A} is a *solution* to a nogood δ (resp. a set Δ of nogoods), iff $\delta \not\subseteq \mathbf{A}$ (resp. $\delta \not\subseteq \mathbf{A}$ for all $\delta \in \Delta$).

2.1 HEX-Programs

HEX-programs were introduced in [11] as a generalization of (disjunctive) extended logic programs under the answer set semantics [16]; for details and background see [11].

Syntax. HEX-programs extend ordinary ASP programs by *external atoms*, which enable a bidirectional interaction between a program and external sources of computation. External atoms have a list of input parameters (constants or predicate names) and a list of output parameters. Informally, to evaluate an external atom, the reasoner passes the constants and extensions of the predicates in the input tuple to the external source associated with the external atom. The external source computes output tuples which are matched with the output list. More formally, a *ground external atom* is of the form

$$\&g[\mathbf{p}](\mathbf{c}), \quad (1)$$

where $\mathbf{p} = p_1, \dots, p_k$ are constant input parameters (predicate names or object constants), and $\mathbf{c} = c_1, \dots, c_l$ are constant output terms.

Ground HEX-programs are then defined similar to ground ordinary ASP programs.

Definition 1 (Ground HEX-programs). A ground HEX-program consists of rules

$$a_1 \vee \dots \vee a_k \leftarrow b_1, \dots, b_m, \text{not } b_{m+1}, \dots, \text{not } b_n, \quad (2)$$

where each a_i is an (ordinary) ground atom $p(c_1, \dots, c_\ell)$ with constants c_i , $1 \leq i \leq \ell$, each b_j is either an ordinary ground atom or a ground external atom, and $k + n > 0$.²

The *head* of a rule r is $H(r) = \{a_1, \dots, a_k\}$ and the *body* is $B(r) = \{b_1, \dots, b_m, \text{not } b_{m+1}, \dots, \text{not } b_n\}$. We call b or $\text{not } b$ in a rule body a *default literal*; $B^+(r) = \{b_1, \dots, b_m\}$ is the *positive body*, $B^-(r) = \{b_{m+1}, \dots, b_n\}$ is the *negative body*. For a program Π , let $A(\Pi)$ be the set of all ordinary atoms occurring in Π .

We also use non-ground programs. However, as suitable safety conditions allow for using a grounding procedure [12], we limit our investigation to ground programs.

Semantics and Evaluation. Intuitively, a ground external atom $\&g[p](c)$ is true, if the external source $\&g$ yields output tuple c when evaluated with input p . Formally, the semantics of a ground external atom $\&g[p](c)$ wrt. an interpretation \mathbf{A} is given by the value of a $1+k+l$ -ary Boolean *oracle function* $f_{\&g}$ that is defined for all possible values of \mathbf{A} , p and c , where k is the length of p and l is the length of c . Thus, $\&g[p](c)$ is true relative to \mathbf{A} if and only if it holds that $f_{\&g}(\mathbf{A}, p, c) = 1$. Satisfaction of ordinary rules and ASP programs [16] is then extended to HEX-rules and programs in the obvious way, and the notion of extension $\text{ext}(\cdot, \mathbf{A})$ for external predicates $\&g$ with input lists p is naturally defined by $\text{ext}(\&g[p], \mathbf{A}) = \{c \mid f_{\&g}(\mathbf{A}, p, c) = 1\}$.

Definition 2 (FLP-Reduct [14]). For an interpretation \mathbf{A} over a program Π , the FLP-reduct $f\Pi^{\mathbf{A}}$ of Π wrt. \mathbf{A} is the set $\{r \in \Pi \mid \mathbf{A} \models b, \text{ for all } b \in B(r)\}$ of all rules whose body is satisfied under \mathbf{A} .

An assignment \mathbf{A}_1 is smaller or equal to another assignment \mathbf{A}_2 wrt. a program Π , denoted $\mathbf{A}_1 \leq_\Pi \mathbf{A}_2$ iff $\{\mathbf{T}a \in \mathbf{A}_1^T \mid a \in A(\Pi)\} \subseteq \{\mathbf{T}a \in \mathbf{A}_2^T \mid a \in A(\Pi)\}$.

Definition 3 (Answer Set). An answer set of Π is a \leq_Π -minimal (complete) model \mathbf{A} of $f\Pi^{\mathbf{A}}$.

Since interpretations (and thus answer sets, etc.) are complete assignments, slightly abusing notation, we adopt the usual convention to uniquely identify them with the set of all positive literals they contain.

Example 1. Consider the program $\Pi = \{p \leftarrow \&id[p]()\}$, where $\&id[p]()$ is true iff p is true. Then Π has the answer set $\mathbf{A}_1 = \emptyset$, which is indeed a \leq_Π -minimal model of $f\Pi^{\mathbf{A}_1} = \emptyset$.

The answer sets of a HEX-program Π are determined by the DLVHEX solver using a transformation to ordinary ASP programs as follows. Each external atom $\&g[p](c)$ in Π is replaced by an ordinary ground *external replacement atom* $e_{\&g[p]}(c)$ and a rule $e_{\&g[p]}(c) \vee ne_{\&g[p]}(c) \leftarrow$ is added to the program. The answer sets of the resulting *guessing program* $\hat{\Pi}$ are determined by an ordinary ASP solver and projected to non-replacement atoms. However, the resulting interpretations are not necessarily models

² For simplicity, we do not formally introduce strong negation but view, as customary, classical literals $\neg a$ as new atoms together with a constraint $\leftarrow a, \neg a$.

of Π , as the value of $\&g[p]$ under $f_{\&g}$ can be different from the one of $e_{\&g[p]}(\mathbf{c})$. Each answer set of $\hat{\Pi}$ is thus merely a *candidate* which must be checked against the external sources. If no discrepancy is found, the model candidate is a *compatible set* of Π . More precisely,

Definition 4 (Compatible Set). A compatible set of a program Π is an interpretation $\hat{\mathbf{A}}$ such that

- (i) $\hat{\mathbf{A}}$ is an answer set [16] of the guessing program $\hat{\Pi}$, and
- (ii) $f_{\&g}(\hat{\mathbf{A}}, \mathbf{p}, \mathbf{c}) = 1$ iff $\mathbf{T}e_{\&g[p]}(\mathbf{c}) \in \hat{\mathbf{A}}$ for all external atoms $\&g[p](\mathbf{c})$ in Π , i.e. the guessed values coincide with the actual output under the input from $\hat{\mathbf{A}}$.

The compatible sets of Π include (modulo $A(\Pi)$) all (FLP) answer sets. For each answer set \mathbf{A} there is a compatible set $\hat{\mathbf{A}}$ such that \mathbf{A} is the restriction of $\hat{\mathbf{A}}$ to non-replacement atoms, but not vice versa. To filter out the compatible sets which are not answer sets, the current evaluation algorithm proceeds as follows. Each compatible set \mathbf{A} is fed to the minimality check, which is realized as a search for unfounded sets. This is justified by the following Definitions 5 and 6 and Theorem 1 from [7]. (These results lift unfounded sets for disjunctive logic programs with arbitrary aggregates [13] to HEX-programs.)

Definition 5 (Unfounded Set [7]). Given a program Π and an interpretation \mathbf{A} , let X be any set of ordinary ground atoms appearing in Π . Then, X is an unfounded set for \mathbf{A} iff, for each rule r having some atoms from X in the head, at least one of the following conditions holds, where $\mathbf{A} \dot{\cup} \neg.X = (\mathbf{A} \setminus \{\mathbf{T}a \mid a \in X\}) \cup \{\mathbf{F}a \mid a \in X\}$:

- (i) some literal of $B(r)$ is false wrt. \mathbf{A} ,
- (ii) some literal of $B(r)$ is false wrt. $\mathbf{A} \dot{\cup} \neg.X$, or
- (iii) some atom of $H(r) \setminus X$ is true wrt. \mathbf{A} .

Definition 6 (Unfounded-free Interpretations [7]). An interpretation \mathbf{A} of a program Π is unfounded-free iff $\mathbf{A}^T \cap X = \emptyset$, for all unfounded sets X of Π wrt. \mathbf{A} .

Theorem 1 (Characterization of Answer Sets [7]). A model \mathbf{A} of a program Π is an answer set iff it is unfounded-free.

Example 2 (cont'd). Reconsider the program $\Pi = \{p \leftarrow \&id[p]()\}$ from above. Then the corresponding guessing program is $\hat{\Pi} = \{p \leftarrow e_{\&id[p]}(); e_{\&id[p]} \vee ne_{\&id[p]} \leftarrow\}$ and has the answer sets $\mathbf{A}_1 = \emptyset$ and $\mathbf{A}_2 = \{\mathbf{T}p, \mathbf{T}e_{\&id[p]}\}$. While \mathbf{A}_1 does not intersect with any unfounded sets and is thus also a \leq_{Π} -minimal model of $f\Pi^{\mathbf{A}_1} = \emptyset$, \mathbf{A}_2 intersects with the unfounded set $U = \{p\}$ and is not an answer set.

Our HEX implementation DLVHEX realizes the search for unfounded sets as a separate search problem using an encoding as a SAT instance. That is, for a program Π and an interpretation \mathbf{A} we construct a set of nogoods $\Gamma_{\Pi}^{\mathbf{A}}$ such that its solutions contain representations of all unfounded sets of Π wrt. \mathbf{A} . A (relatively simple) post-check finds the unfounded sets among the solutions of $\Gamma_{\Pi}^{\mathbf{A}}$.

3 Deciding the Necessity of the UFS Check

An alternative to the search for unfounded sets is an explicit construction of the reduct and a search for smaller models. However, it turned out that the minimality check based

on unfounded sets is more efficient. Nevertheless the computational costs are still high. Moreover, during evaluation of \tilde{I} for computing the compatible set $\tilde{\mathbf{A}}$, the ordinary ASP solver has already made an unfounded set check, and we can safely assume that it is founded from its perspective. Hence, all remaining unfounded sets which were not discovered by the ordinary ASP solver have to involve external sources, as their behavior is not fully captured by the ASP solver.

In this section we formalize these ideas and define a decision criterion which allows us to decide whether a further UFS check is necessary for a given program. We eventually define a class of programs which does not require an additional unfounded set check. Intuitively, we show that every unfounded set that is not already detected during the construction of $\tilde{\mathbf{A}}$ contains input atoms of external atoms which are involved in cycles. If no such input atom exists in the program, then the UFS check is superfluous.

Let us therefore start with a definition of atom dependency.

Definition 7 (Atom Dependency). For a ground program Π , and ground atoms $p(\mathbf{c})$ and $q(\mathbf{d})$, we say that

- (i) $p(\mathbf{c})$ depends on $q(\mathbf{d})$, denoted $p(\mathbf{c}) \rightarrow q(\mathbf{d})$, iff for some rule $r \in \Pi$ we have $p(\mathbf{c}) \in H(r)$ and $q(\mathbf{d}) \in B^+(r)$;
- (ii) $p(\mathbf{c})$ depends externally on $q(\mathbf{d})$, denoted $p(\mathbf{c}) \rightarrow_e q(\mathbf{d})$, iff for some rule $r \in \Pi$ we have $p(\mathbf{c}) \in H(r)$ and there is a $\&g[q_1, \dots, q_n](\mathbf{e}) \in B^+(r) \cup B^-(r)$ with $q_i = q$ for some $1 \leq i \leq n$.

In the following, we consider *dependency graphs* G_{Π}^R for a ground program Π , where the set of vertices is the set of all ground atoms, and the set of edges is given by a binary relation R over ground atoms. If R is not explicitly mentioned, then it is assumed to consist of $\rightarrow \cup \rightarrow_e$, whose elements are also called *ordinary edges* and *e-edges*, respectively.

The next definition and lemma allow to restrict our attention to the “core” of an unfounded set, i.e., its most essential part. For our purpose, we can then focus on such cores, disregarding atoms in a cut which is defined as follows.

Definition 8 (Cut). Let U be an unfounded set of Π wrt. \mathbf{A} . A set of atoms $C \subseteq U$ is called a *cut*, iff

- (i) $b \not\rightarrow_e a$, for all $a \in C$ and $b \in U$ (C has no incoming or internal e-edges), and
- (ii) $b \not\rightarrow a$ and $a \not\rightarrow b$, for all $a \in C$ and $b \in U \setminus C$ (there are no ordinary edges between C and $U \setminus C$).

Example 3. Consider the program Π given as the following set of rules

$$\begin{aligned} r &\leftarrow \&id[r]() \\ p &\leftarrow \&id[r]() \\ p &\leftarrow q \\ q &\leftarrow p \end{aligned}$$

We have $p \rightarrow q$, $q \rightarrow p$, $r \rightarrow_e r$ and $p \rightarrow_e r$. Program Π has the unfounded set $U = \{p, q, r\}$ wrt. $\mathbf{A} = \{\mathbf{T}p, \mathbf{T}q, \mathbf{T}r\}$. Observe that $C = \{p, q\}$ is a cut, and therefore we have that $U \setminus C = \{r\}$ is an unfounded set of Π wrt. \mathbf{A} .

We first prove that cuts can be removed from unfounded sets and the resulting set is still an unfounded set.

Lemma 1 (Unfounded Set Reduction Lemma). *Let U be an unfounded set of Π wrt. \mathbf{A} , and let C be a cut. Then, $Y = U \setminus C$ is an unfounded set of Π wrt. \mathbf{A} .*

Proof (Sketch). If $Y = \emptyset$, then the result holds trivially. Otherwise, let $r \in \Pi$ with $H(r) \cap Y \neq \emptyset$. We show that one of the conditions in Definition 5 holds. Observe that $H(r) \cap U \neq \emptyset$ because $U \supseteq Y$. Since U is an unfounded set of Π wrt. \mathbf{A} , either

- (i) $\mathbf{A} \not\models b$ for some $b \in B(r)$; or
- (ii) $\mathbf{A} \dot{\cup} \neg.U \not\models b$ for some $b \in B(r)$; or
- (iii) $\mathbf{A} \models h$ for some $h \in H(r) \setminus U$

If (i), then the condition also holds wrt. Y .

If (ii), let $a \in H(r)$ such that $a \in Y$, and $b \in B(r)$ such that $\mathbf{A} \dot{\cup} \neg.U \not\models b$. We make a case distinction: either b is an ordinary literal or an external one.

If it is an ordinary default-negated atom not c , then $\mathbf{A} \dot{\cup} \neg.U \not\models b$ implies $\mathbf{T}c \in \mathbf{A}$ and $c \notin U$, and therefore also $\mathbf{A} \dot{\cup} \neg.Y \not\models b$. So assume b is an ordinary atom. If $b \notin U$ then $\mathbf{A} \not\models b$ and case (i) applies, so assume $b \in U$. Because $a \in H(r)$ and $b \in B(r)$, we have $a \rightarrow b$ and therefore either $a, b \in C$ or $a, b \in Y$ (because there are no ordinary edges between C and Y). But by assumption $a \in Y$, and therefore $b \in Y$, hence $\mathbf{A} \dot{\cup} \neg.Y \not\models b$.

If b is an external literal, then there is no $q \in U$ with $a \rightarrow_e q$ and $q \notin Y$. Otherwise, this would imply $q \in C$ and C would have an incoming e-edge, which contradicts the assumption that C is a cut. Hence, for all $q \in U$ with $a \rightarrow_e q$, also $q \in Y$, and therefore the truth value of b under $\mathbf{A} \dot{\cup} \neg.U$ and $\mathbf{A} \dot{\cup} \neg.Y$ is the same. Hence $\mathbf{A} \dot{\cup} \neg.Y \not\models b$.

If (iii), then also $\mathbf{A} \models h$ for some $h \in H(r) \setminus Y$ because $Y \subseteq U$ and therefore $H(r) \setminus Y \subseteq H(r) \setminus U$. \square

Next we prove, intuitively, that for each unfounded set U of Π , either the input to some external atom is unfounded itself, or U is already detected when $\hat{\Pi}$ is evaluated.

Lemma 2 (EA-Input Unfoundedness). *Let U be an unfounded set of Π wrt. \mathbf{A} . If there are no $x, y \in U$ such that $x \rightarrow_e y$, then U is an unfounded set of $\hat{\Pi}$ wrt. $\hat{\mathbf{A}}$.*

Proof (Sketch). If $U = \emptyset$, then the result holds trivially. Otherwise, let $\hat{r} \in \hat{\Pi}$ such that $H(\hat{r}) \cap U \neq \emptyset$. Let $a \in H(\hat{r}) \cap U$. Observe that \hat{r} cannot be an external atom guessing rule because U contains only ordinary atoms. We show that one of the conditions in Definition 5 holds for \hat{r} wrt. $\hat{\mathbf{A}}$.

Because \hat{r} is no external atom guessing rule, there is a corresponding rule $r \in \Pi$ containing external atoms in place of replacement atoms. Because U is an unfounded set of Π and $H(r) = H(\hat{r})$, either:

- (i) $\mathbf{A} \not\models b$ for some $b \in B(r)$; or
- (ii) $\mathbf{A} \dot{\cup} \neg.U \not\models b$ for some $b \in B(r)$; or
- (iii) $\mathbf{A} \models h$ for some $h \in H(r) \setminus U$

If (i), let $b \in B(r)$ such that $\mathbf{A} \not\models b$ and \hat{b} the corresponding literal in $B(\hat{b})$ (which is the same if b is ordinary and the corresponding replacement literal if b is external). Then also $\hat{\mathbf{A}} \not\models \hat{b}$ because $\hat{\mathbf{A}}$ is compatible.

For (ii), we make a case distinction: either b is ordinary or external.

If b is ordinary, then $b \in B(\hat{r})$ and $\hat{\mathbf{A}} \dot{\cup} \neg.U \not\models b$ holds because \mathbf{A} and $\hat{\mathbf{A}}$ are equivalent for ordinary atoms.

If b is an external atom or default-negated external atom, then no atom $p(\mathbf{c}) \in U$ is input to it, i.e. p is not a predicate input parameter of b ; otherwise we had $a \rightarrow_e p(\mathbf{c})$, contradicting our assumption that U has no internal e-edges. But then $\mathbf{A} \dot{\cup} \neg.U$ implies $\mathbf{A} \not\models b$ because the truth value of b under $\mathbf{A} \dot{\cup} \neg.U$ and \mathbf{A} is the same. Therefore we can apply case (i).

If (iii), then also $\hat{\mathbf{A}} \models h$ for some $h \in H(\hat{r}) \setminus U$ because $H(r) = H(\hat{r})$ contains only ordinary atoms and \mathbf{A} is equivalent to $\hat{\mathbf{A}}$ for ordinary atoms. \square

Example 4. Reconsider the program Π from Example 3. Then the unfounded set $U' = \{p, q\}$ wrt. $\mathbf{A}' = \{\mathbf{T}p, \mathbf{T}q, \mathbf{F}r\}$ is already detected when $\hat{\Pi}$ consisting of

$$\begin{aligned} e_{\&id[r]}() \vee ne_{\&id[r]}() &\leftarrow \\ r &\leftarrow e_{\&id[r]}() \\ p &\leftarrow e_{\&id[r]}() \\ p &\leftarrow q \\ q &\leftarrow p \end{aligned}$$

is evaluated by the ordinary ASP solver because $p \not\rightarrow_e q$ and $q \not\rightarrow_e p$. In contrast, the unfounded set $U'' = \{p, q, r\}$ wrt. $\mathbf{A}'' = \{\mathbf{T}p, \mathbf{T}q, \mathbf{T}r\}$ is *not* detected by the ordinary ASP solver because $p, r \in U''$ and $p \rightarrow_e r$.

The essential property of unfounded sets of Π wrt. \mathbf{A} that are not recognized during the evaluation of $\hat{\Pi}$, is the existence of cyclic dependencies including input atoms of some external atom. Towards a formal characterization of a class of programs without this property, i.e., that do not require additional UFS checks, we define cycles as follows.

Definition 9 (Cycle). A cycle under a binary relation \circ is a sequence of elements $C = c_0, c_1, \dots, c_n, c_{n+1}$ with $n \geq 0$, such that $(c_i, c_{i+1}) \in \circ$ for all $0 \leq i \leq n$ and $c_0 = c_{n+1}$. We say that a set S contains a cycle under \circ , if there is a cycle $C = c_0, c_1, \dots, c_n, c_{n+1}$ under \circ such that $c_i \in S$ for all $0 \leq i \leq n+1$.

The following proposition states, intuitively, that each unfounded set U of Π wrt. \mathbf{A} which contains no cycle through the input atoms to some external atom has a corresponding unfounded set U' of $\hat{\Pi}$ wrt. $\hat{\mathbf{A}}$. That is, the unfoundedness is already detected when $\hat{\Pi}$ is evaluated.

Let $\rightarrow^d = \rightarrow \cup \leftarrow \cup \rightarrow_e$, where \leftarrow is the inverse of \rightarrow , i.e. $\leftarrow = \{(x, y) \mid (y, x) \in \rightarrow\}$. A cycle $c_0, c_1, \dots, c_n, c_{n+1}$ under \rightarrow^d is called an *e-cycle*, iff it contains e-edges, i.e., iff $(c_i, c_{i+1}) \in \rightarrow_e$ for some $0 \leq i \leq n$.

Proposition 1 (Relevance of e-cycles). Let $U \neq \emptyset$ be an unfounded set of Π wrt. \mathbf{A} that does not contain any e-cycle under \rightarrow^d . Then, there exists a nonempty unfounded set of $\hat{\Pi}$ wrt. $\hat{\mathbf{A}}$.

Proof (Sketch). We define the *reachable set* $R(a)$ from some atom a as

$$R(a) = \{b \mid (a, b) \in \{\rightarrow \cup \leftarrow\}^*\},$$

i.e. the set of atoms $b \in U$ reachable from a using edges from $\rightarrow \cup \leftarrow$ only but no e-edges.

We first assume that U contains at least one e-edge, i.e. there are $x, y \in U$ such that $x \rightarrow_e y$. Now we show that there is a $u \in U$ with outgoing e-edge (i.e. $u \rightarrow_e v$

for some $v \in U$), but such that $R(u)$ has no incoming e-edges (i.e. for all $v \in R(u)$ and $b \in U$, $b \not\rightarrow_e v$ holds). Suppose to the contrary that for all a with outgoing e-edges, the reachable set $R(a)$ has an incoming e-edge. We now construct an e-cycle under \rightarrow^d , which contradicts our assumption. Start with an arbitrary node with an outgoing e-edge $c_0 \in U$ and let p_0 be the (possibly empty) path (under $\rightarrow \cup \leftarrow$) from c_0 to the node $d_0 \in R(c_0)$ such that d_0 has an incoming e-edge, i.e. there is a c_1 such that $c_1 \rightarrow_e d_0$; note that $c_1 \notin R(c_0)$ ³. By assumption, also some node d_1 in $R(c_1)$ has an incoming e-edge (from some node $c_2 \notin R(c_1)$). Let p_1 be the path from c_1 to d_1 , etc. By iteration we can construct the concatenation of the paths $p_0, (d_0, c_1), p_1, (d_1, c_2), p_2, \dots, p_i, (d_i, c_{i+1}), \dots$, where the p_i from c_i to d_i are the paths within reachable sets, and the (d_i, c_{i+1}) are the e-edges between reachable sets. However, as U is finite some nodes on this path must be equal, i.e., a prefix of the constructed sequence represents an e-cycle (in reverse order).

This proves that u is a node with outgoing e-edge but such that $R(u)$ has no incoming e-edges. We next show that $R(u)$ is a cut. Condition (i) is immediately satisfied by definition of u . Condition (ii) is shown as follows. Let $u' \in R(u)$ and $v' \in U \setminus R(u)$. We have to show that $u' \not\rightarrow v'$ and $v' \not\rightarrow u'$. Suppose, towards a contradiction, that $u' \rightarrow v'$. Because of $u' \in R(u)$, there is a path from u to u' under $\rightarrow \cup \leftarrow$. But if $u' \rightarrow v'$, then there would also be a path from u to v' under $\rightarrow \cup \leftarrow$ and v' would be in $R(u)$, a contradiction. Analogously, $v' \rightarrow u'$ would also imply that there is a path from u to v' because there is a path from u to u' , again a contradiction.

Therefore, $R(u)$ is a cut of U , and by Lemma 1, it follows that $U \setminus R(u)$ is an unfounded set. Observe that $U \setminus R(u)$ contains one e-edge less than U because u has an outgoing e-edge. Further observe that $U \setminus R(u) \neq \emptyset$ because there is a $w \in U$ such that $u \rightarrow_e w$ but $w \notin R(u)$. By iterating this argument, the number of e-edges in the unfounded set can be reduced to zero in a nonempty core. Eventually, Lemma 2 applies, proving that the remaining set is an unfounded set of $\hat{\Pi}$. \square

Corollary 1. *If there is no e-cycle under \rightarrow^d and $\hat{\Pi}$ has no unfounded set wrt. $\hat{\mathbf{A}}$, then \mathbf{A} is unfounded-free for Π .*

Proof (Sketch). Suppose there is an unfounded set U of Π wrt. \mathbf{A} . Then it contains no e-cycle because there is no e-cycle under \rightarrow^d . Then by Proposition 1 there is an unfounded set of $\hat{\Pi}$ wrt. $\hat{\mathbf{A}}$, which contradicts our assumption. \square

This corollary can be used as follows to increase performance of an evaluation algorithm: if there is no cycle under \rightarrow^d containing e-edges, then an explicit unfounded set check is not necessary because the unfounded set check made during evaluation of $\hat{\Pi}$ suffices. Note that this test can be done efficiently (in fact in linear time, similar to deciding stratifiability of an ordinary logic program). Moreover, in practice one can abstract from \rightarrow^d by using analogous relations on the level of predicate symbols instead of atoms. Clearly, if there is no e-cycle in the predicate dependency graph, then there can also be no e-cycle in the atom dependency graph. Hence, the predicate dependency graph can be used to decide whether the unfounded set check can be skipped.

³ Whenever $x \rightarrow_e y$ for $x, y \in U$, then there is no path from x to y under $\rightarrow \cup \leftarrow$, because otherwise we would have an e-cycle under \rightarrow^d .

Example 5. All example programs considered until here require an UFS check, but the program $\Pi = \{out(X) \leftarrow \&diff[set_1, set_2](X)\} \cup F$ does not for any set of facts F , because there is no e-cycle under \rightarrow^d , where $\&diff$ computes the set difference of the extensions of set_1 and set_2 .

Also $\Pi = \{str(Z) \leftarrow dom(Z), str(X), str(Y), not \&concat[X, Y](Z)\}$ (where $\&concat$ takes two constants and computes their string concatenation) does not need such a check; there is a cycle over an external atom, but no e-cycle under \rightarrow^d .

Moreover, the following proposition states that, intuitively, if $\hat{\Pi}$ has no unfounded sets wrt. $\hat{\mathbf{A}}$, then any unfounded set U of Π wrt. \mathbf{A} must contain an atom which is involved in a cycle under \rightarrow^d that has an e-edge.

Definition 10 (Cyclic Input Atoms). For a program Π , an atom a is a cyclic input atom, iff there is an atom b such that $b \rightarrow_e a$ and there is a path from a to b under \rightarrow^d .

Let $CA(\Pi)$ denote the set of all cyclic input atoms of program Π .

Proposition 2 (Unfoundedness of Cyclic Input Atom). Let $U \neq \emptyset$ be an unfounded set of Π wrt. \mathbf{A} such that U does not contain cyclic input atoms. Then, $\hat{\Pi}$ has a nonempty unfounded set wrt. $\hat{\mathbf{A}}$.

Proof (Sketch). If U contains no cyclic input atoms, then all cycles under \rightarrow^d containing e-edges in the atom dependency graph of Π are broken, i.e. U does not contain an e-cycle under \rightarrow^d . Then by Proposition 1 there is an unfounded set of $\hat{\Pi}$ wrt. $\hat{\mathbf{A}}$. \square

Proposition 2 allows for generating the additional nogood $\{\mathbf{F}a \mid a \in CA(\Pi)\}$ and adding it to $\Gamma_{\Pi}^{\mathbf{A}}$. Again, considering predicate symbols instead of atoms is possible to reduce the overhead introduced by the dependency graph.

4 Program Decomposition

It turns out that the usefulness of the decision criterion can be increased by decomposing the program into components, such that the criterion can be applied component-wise. This allows for restricting the unfounded set check to components with e-cycles, whereas e-cycle-free components can be ignored in the check.

Let \mathcal{C} be a partitioning of the ordinary atoms $A(\Pi)$ of Π into subset-maximal strongly connected components under $\rightarrow \cup \rightarrow_e$. We define for each partition $C \in \mathcal{C}$ the subprogram Π_C associated with C as $\Pi_C = \{r \in \Pi \mid H(r) \cap C \neq \emptyset\}$.

We next show that if a program has an unfounded set U wrt. \mathbf{A} , then $U \cap C$ is an unfounded set wrt. \mathbf{A} for the subprogram of some strongly connected component C .

Proposition 3. Let $U \neq \emptyset$ be an unfounded set of Π wrt. \mathbf{A} . Then, for some Π_C with $C \in \mathcal{C}$ it holds that $U \cap C$ is a nonempty unfounded set of Π_C wrt. \mathbf{A} .

Proof (Sketch). Let U be a nonempty unfounded set of Π wrt. \mathbf{A} . Because \mathcal{C} is a decomposition of $A(\Pi)$ into strongly connected components, the *component dependency graph*

$$\langle \mathcal{C}, \{(C_1, C_2) \mid C_1, C_2 \in \mathcal{C}, \exists a_1 \in C_1, a_2 \in C_2 : (a_1, a_2) \in \rightarrow \cup \rightarrow_e\} \rangle$$

is acyclic. Following the hierarchical component dependency graph from the nodes without predecessor components downwards, we can find a “first” component which has a nonempty intersection with U , i.e., there exists a component $C \in \mathcal{C}$ such that $C \cap U \neq \emptyset$ but $C' \cap U = \emptyset$ for all transitive predecessor components C' of C .

We show that $U \cap C$ is an unfounded set of Π_C wrt. \mathbf{A} . Let $r \in \Pi_C$ be a rule such that $H(r) \cap (U \cap C) \neq \emptyset$. We have to show that one of the conditions of Definition 5 holds for r wrt. \mathbf{A} and $U \cap C$.

Because U is an unfounded set of Π wrt. \mathbf{A} and $H(r) \cap (U \cap C) \neq \emptyset$ implies $H(r) \cap U \neq \emptyset$, we know that one of the conditions holds for r wrt. \mathbf{A} and U . If this is condition (i) or (iii), then it trivially holds also wrt. \mathbf{A} and $U \cap C$ because these conditions depend only on the assignment \mathbf{A} , but not on the unfounded set U .

If it is condition (ii), then $\mathbf{A} \dot{\cup} \neg.U \not\models b$ for some (ordinary or external) body literal $b \in B(r)$. We show next that the truth value of all literals in $B(r)$ is the same under $\mathbf{A} \dot{\cup} \neg.U$ and $\mathbf{A} \dot{\cup} \neg.(U \cap C)$, which proves that condition (ii) holds also wrt. \mathbf{A} and $U \cap C$.

If $b = \text{not } a$ for some atom a , then $\mathbf{T}a \in \mathbf{A}$ and $a \notin U$ and consequently $a \notin U \cap C$, hence $\mathbf{A} \dot{\cup} \neg.(U \cap C) \not\models b$. If b is an ordinary atom, then either $\mathbf{F}b \in \mathbf{A}$, which implies immediatly that $\mathbf{A} \dot{\cup} \neg.(U \cap C) \not\models b$, or $b \in U$. But in the latter case b is either in a predecessor component C' of C or in C itself (since $h \rightarrow b$ for all $h \in H(r)$). But since $U \cap C' = \emptyset$ for all predecessor components of C , we know $b \in C$ and therefore $b \in (U \cap C)$, which implies $\mathbf{A} \dot{\cup} \neg.(U \cap C) \not\models b$.

If b is a positive or default-negated external atom, then all input atoms a to b are either in a predecessor component C' of C or in C itself (since $h \rightarrow_e a$ for all $h \in H(r)$). We show with a similar argument as before that the truth value of each input atom a is the same under $\mathbf{A} \dot{\cup} \neg.U$ and $\mathbf{A} \dot{\cup} \neg.(U \cap C)$: if $\mathbf{A} \dot{\cup} \neg.U \models a$, then $\mathbf{T}a \in \mathbf{A}$ and $a \notin U$, hence $a \notin (U \cap C)$ and therefore $\mathbf{A} \dot{\cup} \neg.(U \cap C) \models a$. If $\mathbf{A} \dot{\cup} \neg.U \not\models a$, then either $\mathbf{F}a \in \mathbf{A}$, which immediately implies $\mathbf{A} \dot{\cup} \neg.(U \cap C) \not\models a$, or $a \in U$. But in the latter case a must be in C because $U \cap C' = \emptyset$ for all predecessor components C' of C . Therefore $a \in (U \cap C)$ and consequently $\mathbf{A} \dot{\cup} \neg.(U \cap C) \not\models a$. Because all input atoms a have the same truth value under $\mathbf{A} \dot{\cup} \neg.U$ and $\mathbf{A} \dot{\cup} \neg.(U \cap C)$, the same holds also for the positive or default-negated external atom b itself. \square

This proposition states that a search for unfounded sets can be done independently for the subprograms Π_C for all $C \in \mathcal{C}$. If there exists a global unfounded set, then there exists also one in at least one of the program components. However, we know by Corollary 1 that programs Π without e-cycles cannot contain unfounded sets, which are not already detected when \hat{H} is solved. If we apply this proposition to the subprograms Π_C , we can safely ignore e-cycle-free program components.

Example 6. Reconsider the program Π from Example 3. Then \mathcal{C} contains the components $C_1 = \{p, q\}$ and $C_2 = \{r\}$ and we have $\Pi_{C_1} = \{p \leftarrow \&id[r](); p \leftarrow q; q \leftarrow p\}$ and $\Pi_{C_2} = \{r \leftarrow \&id[r]()\}$. By Proposition 3, each unfounded set of Π wrt. some assignment can also be detected over one of the components. Consider e.g. $U = \{p, q, r\}$ wrt. $\mathbf{A} = \{\mathbf{T}p, \mathbf{T}q, \mathbf{T}r\}$. Then $U \cap \{r\} = \{r\}$ is also an unfounded set of Π_{C_2} wrt. \mathbf{A} .

By separate application of Corollary 1 to the components, we can conclude that there can be no unfounded sets over Π_{C_1} that are not already detected when \hat{H} is evaluated (because it has no e-cycles). Hence, the additional unfounded set check is only necessary

#args	first answer set					all answer sets				
	standard approach		new approach		gain	standard approach		new approach		gain
	timeouts	avg	timeouts	avg		timeouts	avg	timeouts	avg	
5	0	1.09	0	1.07	2.16%	0	1.70	0	1.56	8.44%
6	0	2.40	0	2.30	4.38%	0	4.58	0	3.74	18.42%
7	0	5.58	0	5.33	4.47%	0	15.66	0	11.28	27.95%
8	0	14.26	0	12.74	10.70%	3	71.06	2	39.32	44.66%
9	0	39.82	0	33.57	15.70%	16	174.99	8	106.34	39.23%
10	2	126.54	0	80.00	36.78%	40	278.98	16	214.81	23.00%

Table 1: Argumentation Benchmarks: standard approach means the state-of-the-art approach without decomposition of the UFS check and without elimination of unnecessary checks, times are in seconds, timeout was 300 sec, for each system size there were 50 instances.

for Π_{C_2} . Indeed, the only unfounded set which is not detected when $\hat{\Pi}$ is evaluated is $\{r\}$ of Π_{C_2} wrt. any interpretation $\mathbf{A} \supseteq \{\mathbf{Tr}\}$.

Finally, one can also show that splitting, i.e., the component-wise check for foundedness, does not lead to spurious unfounded sets.

Proposition 4. *If U is an unfounded set of Π_C wrt. \mathbf{A} such that $U \subseteq C$, then U is an unfounded set of Π wrt. \mathbf{A} .*

Proof (Sketch). If $U = \emptyset$, then the result holds trivially. By definition of Π_C we have $H(r) \cap C = \emptyset$ for all $r \in \Pi \setminus \Pi_C$. By precondition of the proposition we have $U \subseteq C$. But then $H(r) \cap U = \emptyset$ for all $r \in \Pi \setminus \Pi_C$ and U is an unfounded set of Π wrt. \mathbf{A} . \square

5 Implementation and Evaluation

For implementing our technique, we integrated CLASP into our prototype system DLVHEX; we use CLASP as an ASP solver for computing compatible sets and as a SAT solver for solving the nogood set of the UFS check. We evaluated the implementation on a Linux server with two 12-core AMD 6176 SE CPUs with 128GB RAM.

Argumentation Benchmarks. In this benchmark we compute ideal set extensions for randomized instances of abstract argumentation frameworks [4] of different sizes. In these instances, the cycles involve usually only small parts of the overall programs, hence the program decomposition is very effective. Table 1 shows results of our experimental evaluation on argumentation benchmark instances; for computing average times, we considered 300 seconds for instances that timed out. The encodings contain a cyclic part with cycles over external atoms, and a cyclic part with cycles that do not contain external atoms. Therefore in these instances our new approach can help in limiting the set of atoms for which unfounded sets must be checked, which explains the significant performance gain due to less time spent in the UFS check.

Multi-Context System Benchmarks. MCSs [1] are a formalism for interlinking knowledge based systems; in [9], *inconsistency explanations (IEs)* for an MCS were defined.

This benchmark computes the IEs, which correspond 1-1 to answer sets of an encoding rich in cycles through external atoms (which evaluate local knowledge base semantics). We use random instances of different topologies created with an available benchmark generator. For the MCS benchmarks we tested 68 consistent and 88 inconsistent MCSs for which we compute inconsistency explanations [9]. This encoding contains saturation over external atoms, where nearly all cycles in the HEX-program contain at least one external atom. Therefore the methods we introduce in this work can only very rarely reduce the set of atoms for which the UFS check needs to be performed.

The benchmark result for MCS instances confirms that the syntactic check we introduce in this paper is very cheap and does not impede performance, even if an instance does not admit a considerable simplification for the UFS check: over all 156 instances, we had an overall runtime of 25357 seconds with the standard approach, and a runtime of 25115 seconds with our new approach; the gain is 242 seconds which is less than one percent speedup (for enumerating all inconsistency explanations) by applying our method. This is a very small gain, and there is no difference in the number of instances that timed out.

Default Reasoning over Description Logics Benchmarks. Another application of HEX-programs is the DL-plugin [10], which integrates description logics ontologies with rules. This allows, for instance, default reasoning over description logic knowledge bases, which is not possible in DL knowledge bases alone. Defaults require cyclic dependencies over external atoms. However, as all such dependencies involve default negated atoms, we have no cycles according to Definition 7, which respects only positive dependencies. Hence, the decision criterion comes to the conclusion that no UFS check is required.

We used variants of the benchmarks presented in [6], which query wines from an ontology and classify them as red or white wines, where a wine is assumed to be white unless the ontology explicitly entails the contrary. In this scenario, the decision criterion eliminates all unfounded set checks. However, as there is only one compatible set per instance, there would be only one unfounded set check anyway, hence the speedup due to the decision criterion is not significant. But the effect of the decision criterion can be increased by slightly modifying the scenario such that there are multiple compatible sets. This can be done, for instance, by nondeterministic default classifications, e.g., if a wine is not Italian, then it is either French or Spanish by default. Our experiments have shown that with a small number of compatible sets, the performance enhancement due to the decision criterion is marginal, but increases with larger numbers of compatible sets. For instance, for 243 compatible sets (and thus 243 unfounded set checks) we could observe a speedup from 13.59 to 12.19 seconds.

6 Conclusion

The evaluation of HEX-programs requires a minimality check of model candidates which is realized as an equivalent search for unfounded sets (UFS). However, this check is computationally costly. Moreover, during construction of the model candidate, the ASP solver used as a backend has already performed a “restricted” form of unfounded set check, i.e., an UFS check over the program \tilde{I} , viewing external atoms as ordinary ones. Hence, it already excludes certain unfounded candidates. Redoing a complete UFS

search is thus a waste of resources, and the goal is to minimize the number of additional foundedness checks.

In this paper we presented a syntactic criterion which can be efficiently tested and allows to decide whether an additional UFS check is necessary for a given program. It turned out that the essential property is the existence of cyclic dependencies of atoms which involve predicate inputs to external atoms. If no such dependencies exist, then there is no need for an additional check, and the check built into the ordinary ASP solver is already sufficient. In further elaboration, we have refined the basic idea by splitting the input program into components. This allows for independent applications of the decision criterion to the different components. Thus, the UFS check is restricted to relevant parts of the program, while it can safely be ignored for other parts.

Related to our work is [3], where a similar program decomposition is used, yet for ordinary programs only. While we consider e-cycles, which are specific for HEX-programs, the interest in [3] is with head-cycles with respect to disjunctive rule heads. In fact, our implementation may be regarded as an extension of the work in [3], since the evaluation of \hat{H} follows their principles of performing UFS checks in case of head-cycles. Note however, that the applied component splitting does not generalize the well-known splitting theorem [18] as we consider only positive dependencies for ordinary atoms.

An interesting issue for further research is to consider refinements of the decision criterion, or alternative criteria. One direction for refinement is to dynamically take the model candidate into account, in addition to the program structure, which intuitively may prune dependencies and thus allow to skip the UFS check even in the presence of (syntactic) e-cycles. Another extension is to exploit additional semantic information on the external atoms, e.g., such as (anti-)monotonicity etc. Moreover, a more extensive experimental analysis is subject of our future work, where case studies may give rise to alternative criteria and further optimizations.

References

1. Brewka, G., Eiter, T.: Equilibria in Heterogeneous Nonmonotonic Multi-Context Systems. In: AAAI'07. pp. 385–390. AAAI Press (2007)
2. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. *Commun. ACM* 54(12), 92–103 (2011)
3. Drescher, C., Gebser, M., Grote, T., Kaufmann, B., König, A., Ostrowski, M., Schaub, T.: Conflict-driven disjunctive answer set solving. In: KR'08. pp. 422–432. AAAI Press (2008)
4. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.* 77(2), 321–357 (1995)
5. Eiter, T., Fink, M., Ianni, G., Krennwallner, T., Schüller, P.: Pushing efficient evaluation of HEX programs by modular decomposition. In: LPNMR'11. pp. 93–106 (2011)
6. Eiter, T., Fink, M., Krennwallner, T., Redl, C.: Conflict-driven ASP solving with external sources. *Theory and Practice of Logic Programming: Special Issue ICLP (2012)*, to appear
7. Eiter, T., Fink, M., Krennwallner, T., Redl, C., Schüller, P.: Exploiting Unfounded Sets for HEX-Program Evaluation. In: JELIA'12. (2012), to appear
8. Eiter, T., Fink, M., Krennwallner, T., Redl, C., Schüller, P.: Improving HEX-Program Evaluation based on Unfounded Sets. Tech. Rep. INFOSYS RR-1843-12-08, Institut für Informationssysteme, Technische Universität Wien, A-1040 Vienna, Austria (Jul 2012)

9. Eiter, T., Fink, M., Schüller, P., Weinzierl, A.: Finding explanations of inconsistency in Multi-Context Systems. In: KR'10. pp. 329–339. AAAI Press (2010)
10. Eiter, T., Ianni, G., Krennwallner, T., Schindlauer, R.: Exploiting conjunctive queries in description logic programs. *Ann. Math. Artif. Intell.* 53(1–4), 115–152 (2008),
11. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer-Set Programming. In: IJCAI'05. pp. 90–96. Professional Book Center (2005)
12. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: Effective Integration of Declarative Rules with External Evaluations for Semantic-Web Reasoning. In: ESWC'06. pp. 273–287. Springer (2006)
13. Faber, W.: Unfounded sets for disjunctive logic programs with arbitrary aggregates. In: LPNMR'05. pp. 40–52. Springer (2005)
14. Faber, W., Leone, N., Pfeifer, G.: Semantics and complexity of recursive aggregates in answer set programming. *Artif. Intell.* 175(1), 278–298 (2011)
15. Gebser, M., Kaufmann, B., Schaub, T.: Conflict-driven answer set solving: From theory to practice. *Artif. Intell.* 187–188, 52–89 (2012)
16. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. *New Generat. Comput.* 9(3–4), 365–386 (1991)
17. Giunchiglia, E., Lierler, Y., Maratea, M.: Answer set programming based on propositional satisfiability. *J. Autom. Reason.* 36(4), 345–377 (2006)
18. Lifschitz, V., Turner, H.: Splitting a logic program. In: ICLP'94. pp. 23–37 (1994)
19. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning. *ACM Trans. Comput. Logic* 7(3), 499–562 (2006)
20. Simons, P., Niemelä, I., Sooininen, T.: Extending and Implementing the Stable Model Semantics. *Artif. Intell.* 138, 181–234 (2002)